Treap

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

mt19937 gen;

void randinit()
{
        srand(time(NULL));
        gen.seed(rand());
}

ll getrand(ll lo, ll hi)
{
        uniform_int_distribution<ll> dist(lo, hi);
        return dist(gen);
}

struct node
{
        int key, prior, size;
        int l, r;
        node() {}
        node(int key) : key(key), prior(getrand(1, 1e9)), size(1), l(0), r(0) {}
        node(int key, int prior) : key(key), prior(prior), size(1), l(0), r(0) {}
};

int node_count = 1;
int root = 0;
vector<node> nodes(1);

int newNode()
{
        nodes.emplace_back();
        return node_count++;
}

int newNode(int key)
{
        nodes.emplace_back(key);
        return node_count++;
}

int newNode(int key, int prior)
{
        nodes.emplace_back(key, prior);
        return node_count++;
}

void update_size(int t)
{
        nodes[t].size = 0;
        int l = nodes[t].l;
```

```
        int r = nodes[t].r;
        if (l) nodes[t].size += nodes[l].size;
        if (r) nodes[t].size += nodes[r].size;
        nodes[t].size++;
}
void merge(int t, int l, int r)
{
        if (l == 0 || r == 0)
        {
                nodes[t] = l ? nodes[l] : nodes[r];
        }
        if (nodes[l].prior > nodes[r].prior)
        {
                merge(nodes[l].r, nodes[l].r, r);
                nodes[t] = nodes[l];
        }
        else
        {
                merge(nodes[r].l, l, nodes[r].l);
                nodes[t] = nodes[r];
        }
        update_size(t);
}


void split(int t, int key, int& l, int& r)
{
        if (t == 0)
        {
                l = 0;
                r = 0;
        }
        else if (nodes[t].key <= key)
        {
                l = t;
                split(nodes[t].r, key, nodes[l].r, r);
        }
        else
        {
                r = t;
                split(nodes[t].l, key, l, nodes[r].l);
        }
        if(l) update_size(l);
        if(r) update_size(r);
}

void insert(int& t, int k)
{
        if (t == 0)
        {
                t = k;
        }
        else if (nodes[t].prior < nodes[k].prior)
        {
                split(t, nodes[k].key, nodes[k].l, nodes[k].r);
                t = k;
        }
```

```
        else if (nodes[t].key < nodes[k].key)
        {
                insert(nodes[t].r, k);
        }
        else
        {
                insert(nodes[t].l, k);
        }
        update_size(t);
}

void insert(int x)
{
        int k = newNode(x);
        insert(root, k);
}

void erase(int key, int& t = root)
{
        if (t == 0) return;
        else if (nodes[t].key < key)
        {
                erase(key, nodes[t].r);
        }
        else if (nodes[t].key > key)
        {
                erase(key, nodes[t].l);
        }
        else
        {
                merge(t, nodes[t].l, nodes[t].r);
        }
        update_size(t);
}

int find_kth(int k, int t = root)
{
        int lsize;
        if (nodes[t].l == 0) lsize = 0;
        else lsize = nodes[nodes[t].l].size;
        if (lsize == k) return nodes[t].key;
        else if (lsize < k)
        {
                return find_kth(k - lsize - 1, nodes[t].r);
        }
        else
        {
                return find_kth(k, nodes[t].l);
        }
}
```

KMP

```
vector<int> prefix_function(string s) {
        int n = (int)s.length();
```

```
        vector<int> pi(n);
        for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
                j = pi[j-1];
        if (s[i] == s[j])
                j++;
        pi[i] = j;
        }
        return pi;
}
```

Z-Function

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

Suffix array

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
```

```
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
```

LCP array

```
int lcp(int i, int j) {
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (c[k][i % n] == c[k][j % n]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}
```

Edmonds Karp

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
```

```
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

Matching na dwudzielnym

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
```

```
        }
    }
    return false;
}

int main() {
    //... reading the graph ...

    mt.assign(k, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

LIS

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int l = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (d[l-1] < a[i] && a[i] < d[l])
            d[l] = a[i];
    }

    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;
}
```

Fenwick

```
struct FenwickTree {
    vector<int> bit;  // binary indexed tree
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }
```

```
    FenwickTree(vector<int> const &a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

Kolejka maksimow

```
deque<pair<int, int> > kolejka;
void push(int t[], int a){
    while (!kolejka.empty() && kolejka.back().first <= t[a])
        kolejka.pop_back();
    kolejka.push_back(make_pair(t[a], a));
}

void pop(int t[], int a){
    if (kolejka.front().second == a)
        kolejka.pop_front();
}

void get_maximum() {
    if (kolejka.size() == 0)
        return -1;
    return kolejka.front().first;
}
```

FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;
```

```
    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}
```

FFT Wielomiany

```
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

NTT Wielomiany

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;

void fft(vector<int> & a, bool invert) {
    int n = a.size();
```

```
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
```

```
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};
bool MillerRabin(u64 n) { // returns true if n is prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

Pollard Rho

```
long long mult(long long a, long long b, long long mod) {
    return (__int128)a * b % mod;
}

long long f(long long x, long long c, long long mod) {
    return (mult(x, x, mod) + c) % mod;
}

long long rho(long long n, long long x0=2, long long c=1) {
    long long x = x0;
    long long y = x0;
    long long g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(y, c, n);
        y = f(y, c, n);
        g = gcd(abs(x - y), n);
    }
    return g;
}
```

HLD

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}

void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}
int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```